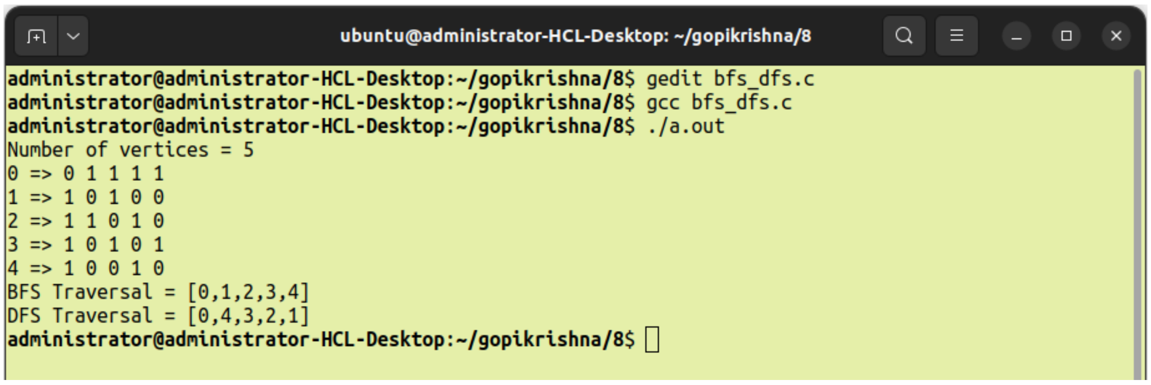
```
/*
            GOPIKRISHNA V
            S3 CSA
            52
*/
#include <stdio.h>
#include <stdlib.h>
#define V  5

void init(int arr[][V])
{
   int i, j;
   for (i = 0; i < V; i++)
      for (j = 0; j < V; j++)
         arr[i][j] = 0;
}

void insertEdge(int arr[][V], int i, int j)
{
   arr[i][j] = 1;
   arr[j][i] = 1;
}

void printAdjMatrix(int arr[][V])
{
   int i, j;
   for (i = 0; i < V; i++)
   {
      printf("%d => ", i);
      for (j = 0; j < V; j++)
      {
         printf("%d ", arr[i][j]);
      }
      printf("\n");
   }
}

struct Node
{
   int key;
   struct Node* next;
};

struct Node* newNode(int k)
{
   struct Node* temp = (struct Node*)malloc(sizeof(struct Node));
   temp->key = k;
   temp->next = NULL;
```

```
    return temp;
}

struct Queue
{
    struct Node *front, *rear, *ptr;
};

struct Queue* createQueue()
{
    struct Queue* q = (struct Queue*)malloc(sizeof(struct Queue));
    q->front = q->rear = q->ptr = NULL;
    return q;
}

void enQueue(struct Queue* q, int k)
{
    struct Node* temp = newNode(k);
    if (q->rear == NULL)
    {
        q->front = q->rear = temp;
        return;
    }
    (q->rear)->next = temp;
    q->rear = temp;
}

int deQueue(struct Queue* q)
{
    if (q->front == NULL)
        return 0;
    struct Node* temp = q->front;
    q->front = q->front->next;
    if (q->front == NULL)
        q->rear = NULL;

    return temp->key;
}

int QisEmpty(struct Queue* q)
{
    return(q->rear == NULL);
}

void Qdisplay(struct Queue* q)
{
    struct Queue* temp = q;
    temp -> ptr = temp -> front;
    if (temp -> ptr == NULL)
    {
        printf("Empty Queue\n");
        return;
    }
    printf("[");
    while((temp->ptr) != NULL)
    {
        printf("%d,",temp->ptr-> key);
        temp->ptr = temp->ptr->next;
    }
    printf("\b]\n");
}

int Qexists(struct Queue* q, int e)
{
    struct Queue* temp = q;
    temp -> ptr = q -> front;
    while((temp->ptr) != NULL)
    {
        if ((temp->ptr)->key == e)
        {
            return 1;
        }
        temp->ptr=temp->ptr->next;
    }
    return 0;
}

void reset(struct Queue* q)
{
    q -> front = q -> rear = NULL;
}

struct Stack
{
    struct Node *top, *ptr;
};

struct Stack* createStack()
{
    struct Stack* s = (struct Stack*)malloc(sizeof(struct Stack));
    s->top = s->ptr = NULL;
    return s;
}

void push(struct Stack* s, int k)
{
    struct Node* temp = newNode(k);
    temp -> next = s -> top;
    s -> top = temp;
}

int pop(struct Stack* s)
{
    if (s -> top == NULL)
        return 0;
    struct Node* temp = s -> top;
    s->top = s->top->next;
    return temp -> key;
}

void Sdisplay(struct Stack* s)
{
    struct Stack* temp = s;
    temp -> ptr = temp -> top;
    if (temp -> ptr == NULL)
    {
        printf("Empty Stack\n");
        return;
    }
    printf("[");
    while((temp->ptr) != NULL)
    {
        printf("%d,",temp->ptr-> key);
        temp->ptr = temp->ptr->next;
    }
    printf("\b]\n");
}

int S_exists(struct Stack* s, int e)
{
    struct Stack* temp = s;
    temp -> ptr = s -> top;

    while((temp->ptr) != NULL)
    {
        if ((temp->ptr)->key == e)
        {
            return 1;
        }
        temp->ptr=temp->ptr->next;
    }
    return 0;
}
```

```
int SisEmpty(struct Stack* s)
{
   return(s->top == NULL);
}

void BFS(int arr[][V], int e,struct Queue* queue, struct Queue* visited)
{
   int i;
   enQueue(visited, e);
   for (i = 0; i < V; i++){
      if ((arr[e][i] == 1) && !((Qexists(visited,i)) || (Qexists(queue,i))))
      {
         enQueue(queue,i);
      }
   }
   if (QisEmpty(queue))
      return;
   e = deQueue(queue);
   BFS(arr,e,queue,visited);
}

void DFS(int arr[][V], int e,struct Stack* stack, struct Queue* visited)
{
   int i;
   enQueue(visited, e);
   for(i=0; i<V; i++)
   {
      if (arr[e][i] == 1 && !((Qexists(visited, i)) || (S_exists(stack, i))))
      {
         push(stack,i);
      }
   }
   if(SisEmpty(stack))
      return;
   e = pop(stack);
   DFS(arr,e,stack,visited);
}

void main()
{
   printf("Number of vertices = %d\n",V);
   int adjMatrix[V][V];

   init(adjMatrix);
   insertEdge(adjMatrix, 0, 1);
   insertEdge(adjMatrix, 0, 2);
   insertEdge(adjMatrix, 0, 3);
   insertEdge(adjMatrix, 1, 2);
```

```
    insertEdge(adjMatrix, 2, 3);
    insertEdge(adjMatrix, 0, 4);
    insertEdge(adjMatrix, 3, 4);

    printAdjMatrix(adjMatrix);

    struct Queue* queue = createQueue();
    struct Queue* visited = createQueue();

    BFS(adjMatrix,0,queue,visited);
    printf("BFS Traversal = ");
    Qdisplay(visited);

    struct Stack* stack = createStack();
    reset(visited);

    DFS(adjMatrix,0,stack,visited);
    printf("DFS Traversal = ");
    Qdisplay(visited);
}
```

OUTPUT



```
ubuntu@administrator-HCL-Desktop: ~/gopikrishna/8
administrator@administrator-HCL-Desktop:~/gopikrishna/8$ gedit bfs_dfs.c
administrator@administrator-HCL-Desktop:~/gopikrishna/8$ gcc bfs_dfs.c
administrator@administrator-HCL-Desktop:~/gopikrishna/8$ ./a.out
Number of vertices = 5
0 => 0 1 1 1 1
1 => 1 0 1 0 0
2 => 1 1 0 1 0
3 => 1 0 1 0 1
4 => 1 0 0 1 0
BFS Traversal = [0,1,2,3,4]
DFS Traversal = [0,4,3,2,1]
administrator@administrator-HCL-Desktop:~/gopikrishna/8$
```